

W5300 Linux Driver Porting Guide

(Version 1.0)



©2009 WIZnet Co., Ltd. All Rights Reserved.

For more information, visit our website at <http://www.wiznet.co.kr>

WIZnet's Online Technical Support

If you have any question or recommendation about our products, please write down

it on Q&A Board in WIZnet website(www.wiznet.co.kr). WIZnet's engineer will give you answer as soon as possible.

The screenshot shows the WIZnet website homepage. At the top, there is a navigation bar with links for HOME, LOGIN, JOIN, CONTACT US, and language options (ENGLISH, CHINESE, JAPANESE, KOREAN). A search bar for 'On-line Mail' is also present. On the left, a vertical menu lists '제품소개', '기술소개', 'Q&A', '자료실', '국내대리', and '파트너사', with 'Q&A' selected and a 'CLICK' button below it. The main content area features a large banner for the W5300 Ethernet chip, highlighting 'Stable 70Mbps Guaranteed (in DMA)!' and 'W5300'. To the right, the WIZ100SR Serial-To-Ethernet Gateway Module is shown with a numbered sequence (01, 02, 03). Below the banner, there are three main sections: 'RoHS Semiconductor Production Line Control System', 'NEW PRODUCT W5300' (listing features like 50 Mbps, 16/8 bit data bus width, 8 independent sockets, and SWTCP/IP support), and 'WHAT'S NEWS' (listing recent news items with dates). The footer includes 'COMPANY OVERVIEW', 'DISTRIBUTOR', and various event announcements like 'WIZnet 3rd Party e-market place', 'WIZnet e-sale Korea', 'WIZnet Ethernet 2007 Winners Announcement', and 'IIC Taiwan Sep. 9-11, 2008 Booth #: 2L06'.

COPYRIGHT NOTICE

Copyright 2009 WIZnet Co., Ltd. All Rights Reserved.

Technical Support: support@wiznet.co.kr

Sales & Distribution: sales@wiznet.co.kr

For more information, visit our website at <http://www.wiznet.co.kr>

Document History Information

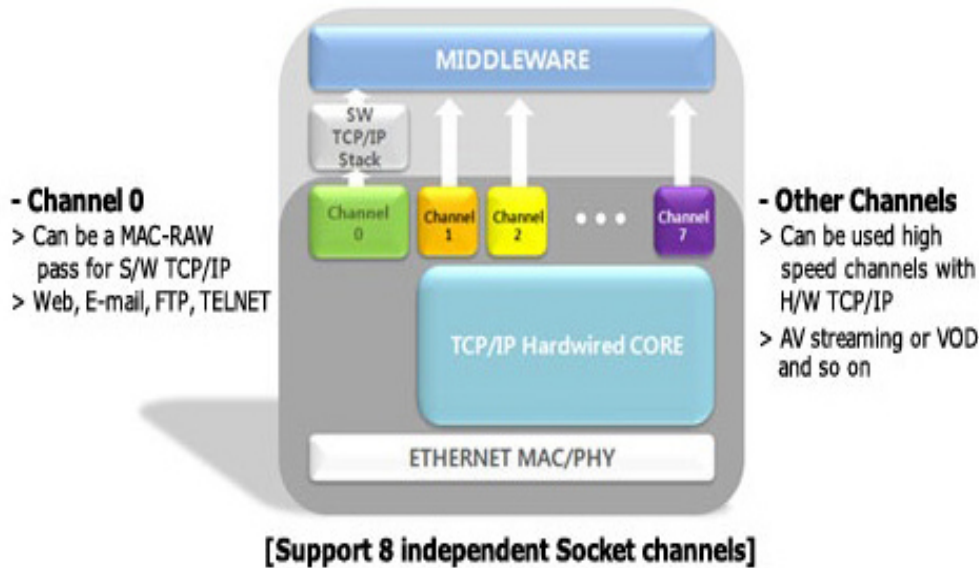
Version	Date	Descriptions
Ver. 1.0.0		First Release

Contents

1. INTRODUCTION.....	6
2. SYSTEM CONFIGURATION	6
2.1. BUS WIDTH CONFIGURATION	7
2.2. BUS ACCESS TIMING CONFIGURATION.....	7
2.3. BASE ADDRESS CONFIGURATION - MAPPING	10
2.4. INTERRUPT PIN CONFIGURATION	10
3. DRIVER PORTING	10
3.1. MAKEFILE	11
3.2. BASE ADDRESS CONFIGURATION	11
3.3. IRQ(INTERRUPT REQUEST) CONFIGURATION.....	11
3.4. MAC ADDRESS	11
3.5. SOCKET RX/TX BUFFER.....	12
3.6. OTHER LINUX KERNEL VERSION	12
4. QUICK START - LOOPBACK PROGRAM	13

1. Introduction

The W5300 Linux driver is for the products which use embedded Linux OS. The W5300 can originally use software protocol stack in OS kernel because it supports conventional *MAC-RAW mode* and also it can use hardware TCP/IP stack simultaneously. We call this mode '*Hybrid mode*'. The *Hardware TCP/IP socket* has same meaning with *Channel* in <Fig 1>



< Fig 1. W5300 Hybrid mode >

This document is porting guide of Linux driver for new system. So it can be divided to three parts as below.

- Bus configuration
- Interrupt configuration
- W5300 Default configuration

In this guide, the examples of system settings are based on W5300E01-ARM board.

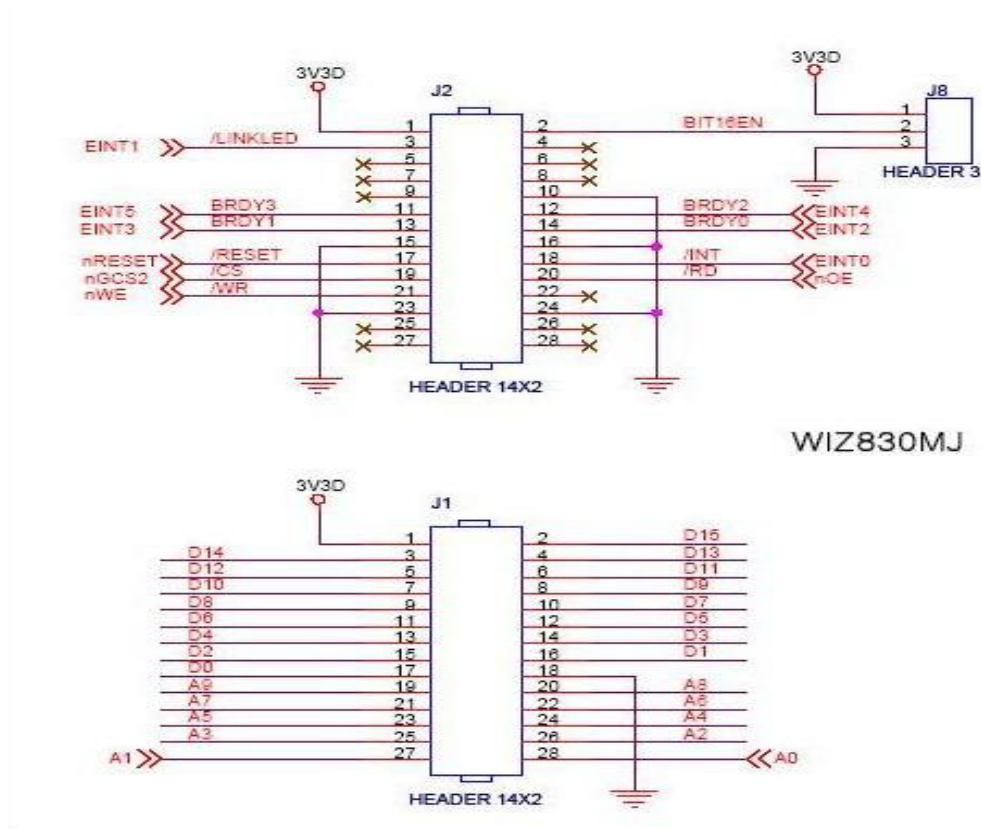
2. System configuration

If the circuit settings of W5300 are right, the MCU setting must be changed according to its hardware. The circuit settings can refer to the document '*6. External Interface*' of '*High-performance Internet Connectivity Solution-W5300*'. In this section, the system setting is generally set up at the bootloader and the kernel.

2.1. Bus Width configuration

The Bus Width which connected from MCU to W5300 must be sets to 16bit. In general, the setting of MCU Bus sets in the initialize routine of Bootloader. Please confirm the settings about MCU Bus from the MCU datasheet.

For example, the S3C2410 which is the MCU of W5300E01-ARM sets the Bus width by using *BWSCON(0x48000000)* register. At the W5300E01-ARM board, the CS pin of W5300 is connected to CS2 pin<Fig 2>. In the *BWSCON* register, the bus width of CS2 sets to 16bit.

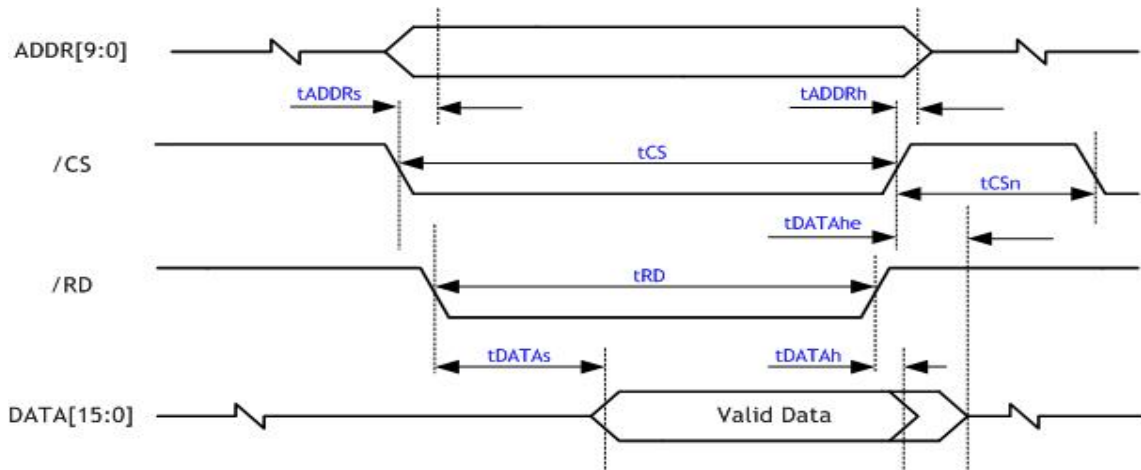


< Fig 2. A part circuit diagram of W5300E01-ARM >

2.2. Bus access timing configuration

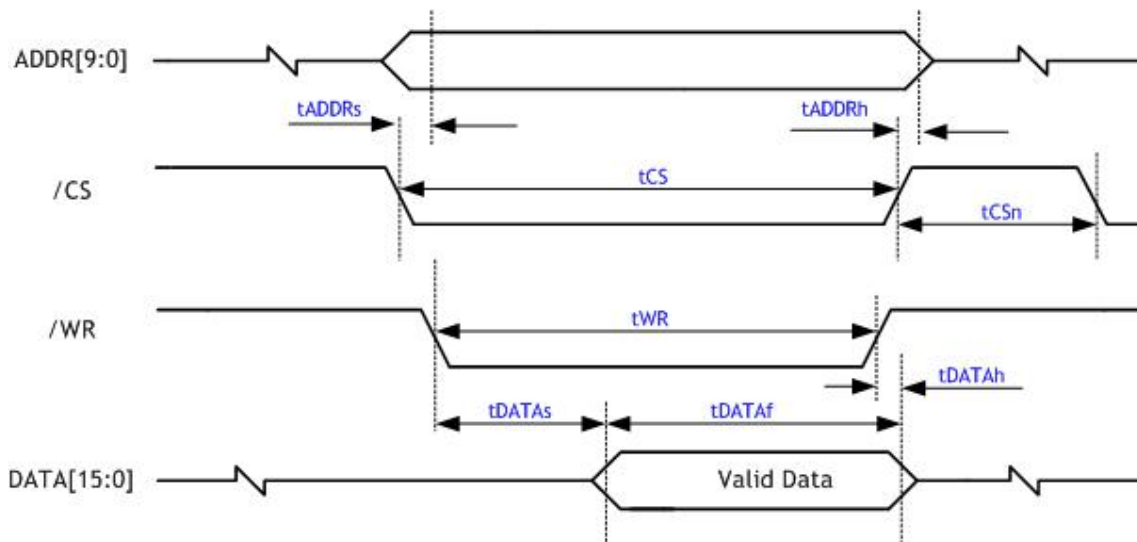
Please set the Bus access timing which access from MCU to W5300.

The */CS low time* of access timing in W5300 has been set to 65ns when read timing and it has been set to 50ns when write timing. (Refer to the '*High-performance Internet Connectivity Solution - W5300*', '*7 Electrical Specifications*')



Description		Min	Max
tADDRs	Address Setup Time after /CS and /RD low	-	7 ns
tADDRh	Address Hold Time after /CS or /RD high	-	-
tCS	/CS Low Time	65 ns	-
tCSn	/CS Next Assert Time	28 ns	-
tRD	/RD Low Time	65 ns	-
tDATAs	DATA Setup Time after /RD low	42 ns	-
tDATAh	DATA Hold Time after /RD and /CS high	-	7 ns
tDATAhe	DATA Hold Extension Time after /CS high	-	2XPLL_CLK

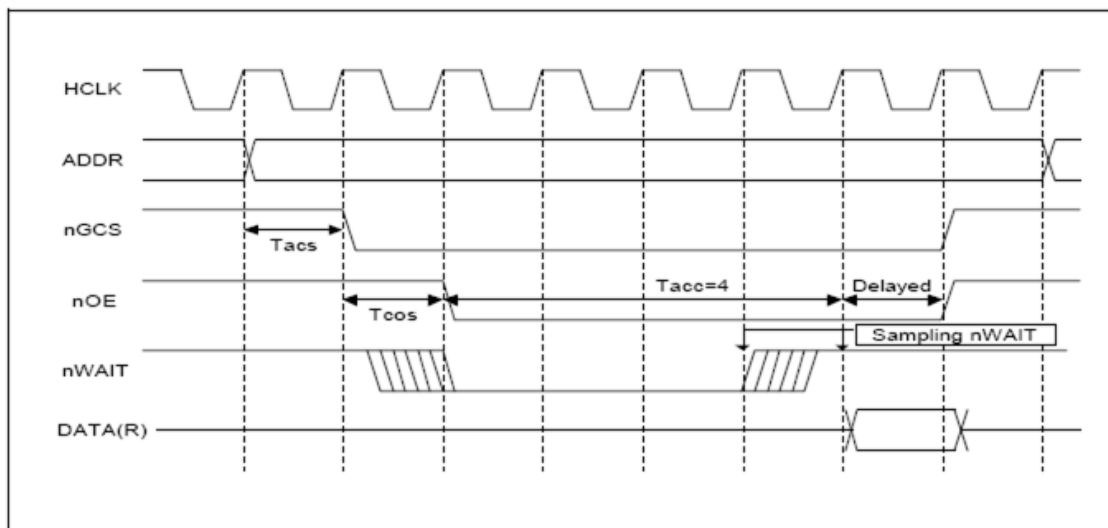
< Fig 3. W5300 READ Timing >



Description		Min	Max
tADDRs	Address Setup Time after /CS and /WR low	-	7 ns
tADDRh	Address Hold Time after /CS or /RD high	-	-
tCS	/CS low Time	50 ns	-
tCSn	/CS next Assert Time	28 ns	-
tWR	/WR low time	50 ns	-
tDATAs	Data Setup Time after /WR low	7 ns	7ns + 7XPLL_CLK
tDATAf	Data Fetch Time	14 ns	tWR-tDATAs
tDATAh	Data Hold Time after /WR high	7 ns	-

< Fig 4. W5300 WRITE Timing >

The Bus access timing of MCU should be set suitably. If the access cycle time sets shorter than 65ns, the data is broken. However the access cycle time sets too long, it comes to be the low data access speed of W5300. So we must set the access cycle time to a little longer than 65ns but the closest time to 65ns.



< Fig 5. W5300E01-ARM BUS Access Timing >

S3C2410 which is the MCU of W5300E01-ARM sets the access timing by using TACS/TCOS/TACC/TACP/TCOH/TCAH registers. Where, register of access cycle setting is TACC. Therefore TACC sets according to previous condition and other registers sets '0'. Since the bus clock of W5300E01-ARM is 100 MHz, the suitable access cycle clock for previous condition is '7'. But it disallows in S3C2410. So we should set the access cycle clock to '8'. (TACC = 5)

2.3. Base address configuration - Mapping

In the Linux kernel, it should be added a routine which is mapping the base address (physical address) of W5300 from driver to virtual address. Generally a hardware dependant routine in the Linux kernel is realized in *'arch/<architecture>/<MCU>/mach-<board>.c'*.

So the W5300E01-ARM contains hardware dependant routine in *'arch/arm/mach-s3c2410/mach-w5300e01.c'*.

```
static struct map_desc w5300e01_iodesc[] __initdata = {
    { 0xf0000000, __phys_to_pfn(S3C2410_CS2), SZ_1M, MT_DEVICE },
    { 0xf8000000, __phys_to_pfn(S3C2410_CS3), SZ_1M, MT_DEVICE }
};
```

Previous code is a routine for mapping the physical address of CS2 to virtual address '0xf0000000'. The physical address of CS2 is base address of W5300 and W5300 Linux driver uses mapped virtual address '0xf0000000'..

2.4. Interrupt pin configuration

In general, we can use GPIO pin to input, output or especially to interrupt by setting it. Namely we can use it variously. So we should set the GPIO pin to interrupt mode.

For example, the GPIO pin set to interrupt in the *'arch/arm/mach-s3c2410/mach-w5300e01.c'* file which is Linux kernel source of W5300E01-ARM as below.

```
static void __init w5300e01_init(void)
{
    ...
    /* W5300 interrupt pin */
    s3c2410_gpio_cfgpin(S3C2410_GPF0, S3C2410_GPF0_EINT0);
    ...
}
```

3. Driver porting

If the system setting is completed in bootloader and Linux kernel, we should do the Linux driver porting of W5300. The compiling of Linux driver needs Linux kernel code in your system. This section serves modification of Linux driver code.

3.1. Makefile

Assign the route of Linux kernel code which we using now. Linux kernel code on the route must be compiled already.

```
# Makefile for w5300 linux driver.  
#  
# linux kernel source path.  
KERNELPATH:=/usr/src/linux  
  
# build options.  
LDFLAGS:=
```

3.2. Base address configuration

Set the virtual address which is assigned from '2.3 Base address configuration - Mapping' to W5300 driver. And the 'w5300.h' of Linux driver code file converts into virtual address which is mapping to base address of W5300 as below.

```
/* FIXME: This value is dependent on the system. It should be modified according  
to system requirement. */  
#define W5300_REG_BASE 0xF0000000
```

3.3. IRQ(Interrupt Request) configuration

Set IRQ number of W5300 interrupt pin to driver. Open the 'w5300.c' file of W5300 driver code and modifies the IRQ number.

```
/* FIXME: Setting irq number / Configurable as module parameter */  
static int w5300_irq = IRQ_EINT0;
```

3.4. Mac address

In the Linux, it is possible that MAC address of W5300 can active change to 'ifconfig' utility. Basically the MAC address is in the 'w5300.c' file.

```
/* FIXME: Setting basic MAC address / It should be set according to the system. */  
const u8 w5300_defmac[MAX_SOCKET_NUM] = {0x00, 0x08, 0xDC,  
0xA0, 0x00, 0x01};
```

3.5. Socket Rx/TX buffer

The W5300 has 8 channels from 0 to 7. The channel 0 operates as MAC RAW mode and supports the operation of conventional MAC chip. And other channels 1~7 support hardware TCP/IP stack. The W5300 can set the Rx/Tx buffer size of each channel respectively. The total buffer size is 128 Kbyte. Of course the more buffer size of channel, the better performance. In the W5300 Linux driver, basic buffer setting is in the 'w5300.c' file. The buffer setting is depended on the specific of its product.

```
/* FIXME: Configuring the size of basic RX/TX FIFO / It should be configured
according to the system */
const u8 w5300_rxbuf_conf[MAX_SOCKET_NUM] = {32, 8, 8, 8, 8, 0, 0, 0};
const u8 w5300_txbuf_conf[MAX_SOCKET_NUM] = {32, 8, 8, 8, 8, 0, 0, 0};
```

<Note> Since channel 0 using MAC RAW mode activated at the MAC layer, it has many received packets. So it is strongly recommended that assigns as large size as possible to the Rx buffer of channel 0.

3.6. Other linux kernel version

The W5300 Linux driver is based on the Linux kernel 2.6.24.4 version which is porting to W5300E01-ARM. In case of using other Linux kernel, it should be modified some different part according to its version.

For example, some functions about NAPI (New API) using in W5300 Linux driver are recently introduced. So in case of using the past Linux kernel version, delete the functions about NAPI and modify the related routine. The difference of NAPI initialize routine is as below.

```
/* Initialization Function of W5300 driver */
static int wiz_init(void)
{
    ...
    /* Setting napi. Enabling to process max 16 packets at a time. */
    netif_napi_add(dev, &wp->napi, wiz_rx_poll, 16);
    ...
}
```

In the past Linux kernel uses the previous routine as below

```
/* Initialization Function of W5300 driver */
static int wiz_init(void)
{
    ...
    /* Setting napi. Enabling to process max 16 packets at a time. */
    dev->poll = wiz_rx_poll;
    dev->weight = 16;
    ...
}
```

Also the `wiz_rx_poll()` function use in NAPI is changed as follows. The type of `wiz_rx_poll` function which we use now is as follows.

```
static int wiz_rx_poll(struct napi_struct *napi, int budget);
```

The previous version of poll function type in the Linux kernel is as follows.

```
static int wiz_rx_poll(struct net_device *dev, int *budget);
```

4. Quick start - Loopback program

In the hybrid mode of W5300, we can use H/W TCP/IP stack of WIZnet supporting same performance of conventional network chip simultaneously. In the hybrid mode, the channel 0 conducts as MAC_RAW mode. It is possible to originally use S/W network stack in the Linux kernel. If we use H/W TCP/IP stack of W5300, just change the socket family at the application.

Add `PF_WIZNET` to '`include/linux/socket.h`' file of Linux kernel as follows.

```
...
#define AF_RXRPC      33      /* RxRPC sockets          */
#define AF_WIZNET    34
#define AF_MAX       35      /* For now.. */
...
#define PF_RXRPC      AF_RXRPC
#define PF_WIZNET     AF_WIZNET
#define PF_MAX        AF_MAX
```

Also we should add PF_WIZNET to header file using by application.

Add PF_WIZNET to `'/usr/include/bits/socket.h'` file.

```
...
#define PF_RRPC      33      /* RRPC sockets.  */
#define PF_WIZNET   34
#define PF_MAX      35      /* For now..  */
...
#define AF_RRPC      PF_RRPC
#define AF_WIZNET   PF_WIZNET
#define AF_MAX      PF_MAX
```

If we want use it without PF_WIZNET in the Linux kernel, we use disabled socket family as a PF_WIZNET by redefinition. Basically default value redefines the PF_BLUETOOTH. We should modify the driver code of `'wizsock.c'` file when we want redefinition with other socket family.

```
/* If PF_WIZNET is not defined, PF_BLUETOOTH is re-defined as PF_WIZNET.
 * In this case, PF_BLUETOOTH is not in use. */
#ifndef PF_WIZNET
#define PF_WIZNET PF_BLUETOOTH
#endif
```

A simple loopback example code by using H/W TCP/IP stack as follows

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
unsigned char data_buf[4096];
#ifndef PF_WIZNET
#define PF_WIZNET PF_BLUETOOTH
#endif
int main(int argc, const char *argv[])
{
    int sd, fd, addr_len, ret;
    struct sockaddr_in sock_in;
```

```
struct sockaddr addr;
// socket open
sd = socket(PF_WIZNET, SOCK_STREAM, 0);
// initialize sockaddr
sock_in.sin_family = PF_WIZNET;
sock_in.sin_addr.s_addr = htonl(INADDR_ANY);
sock_in.sin_port = htons(5300);
addr_len = sizeof(addr);
fd = accept(sd, &addr, &addr_len);
// recv & send (loopback)
while(1) {
    ret = recv(fd, data_buf, 4096, 0);
    if(ret < 0) break;
    ret = send(fd, data_buf, ret, 0);
    if(ret < 0) break;
}
close(fd); // close socket
return 0;
}
```